BioE/MCB/PMB C146/246, Spring 2003

Problem Set 1

Due 27 Jan 02, 5:00 pm PST by email to derek@rana.lbl.gov

1. 1 point
   A. Give 2 examples of different animal anatomical features that are homologous
   B. Give 2 examples of different animal anatomical features that are analogous

   Answers may vary.

2. 2 points

Gene A is in human, and Gene B is in gorilla. Both are descended from a single gene C, in the most recent common ancestor to humans and gorillas. What term(s) describe the evolutionary relationships of these genes? What term(s) do not describe the relationship between these genes?

Genes A & B are homologs; specifically they are orthologs.
Genes A & B are NOT paralogs (they did not arise from a gene duplication event), nor analogs.

3. 2 points

You are given the following four gene sequences:

Human	`AATATTCAAGCGCTACCTATA`
Gorilla	`AATAATGAAGCGCTACCTATA`
Mouse	`AATTATCAAACGGTGCCTACA`
Rat	`ATTTATCAAGCGGTGCCTACA`

Describe how a tree could be used to determine whether these genes are homologous. Do you think they are homologous? Use sample trees to support your argument, but a quantitative proof is not required.

For genes to be homologous, they must have evolved from a common ancestor. A tree for the given genes can be inferred by parsimony: relationships between genes are arranged such that there are a minimum number of nucleotide changes. The nucleotide sequence of the most common ancestor of a given set of genes can be inferred from the extant sequences. Finally, a statistical test is performed to distinguish between homologous characters (divergent evolution) and analogous evolution (convergent evolution). If the number of mutations between the ancestral sequences is fewer than the number predicted by a background mutation model, then we infer that divergent evolution has occurred and the genes are homologous.

These sequences seem to be homologous, because the ancestral sequences of the primate and rodent branches have fewer mutations than the extant sequences. The gene tree also agrees with the species tree. Further statistical tests need to be conducted.

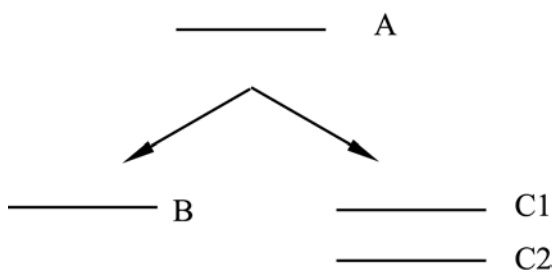
4. 1 point

A. Is homology transitive? (i.e., if A is homologous to B and C, then are B and C homologous?) Why?

B. Is orthology transitive? Why?

Homology is TRANSITIVE. If gene A and gene B both share a common ancestor, and if gene A and C both share a common ancestor, we can assume that genes A, B and C all share a common ancestor and are thus homologous. However, this reasoning ignores the case of multidomain proteins that do not have a simple evolutionary tree.

Orthology is NOT transitive. Consider a counterexample from the diagram below. Gene B is orthologous to gene C1, as they share a common ancestor (A). Gene B is also orthologous to gene C2, as they share the same common ancestor. However, since a gene duplication event likely occurred in the right branch, gene C1 and gene C2 are paralogs.

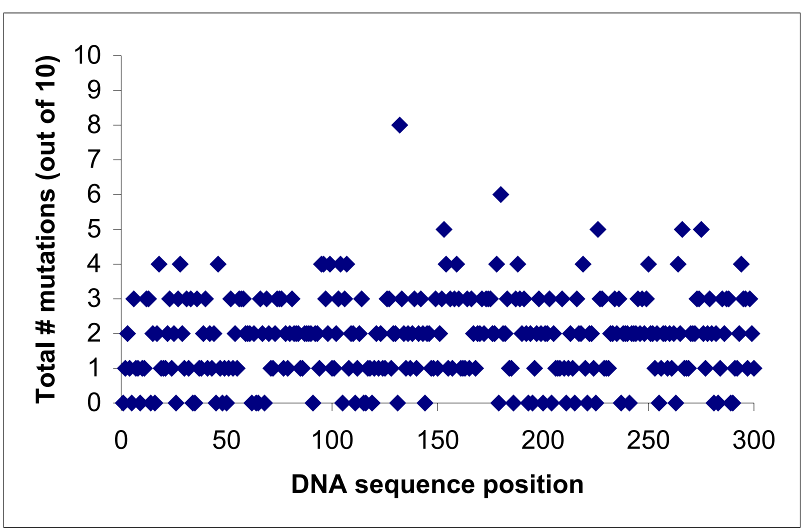


5. 4 points

Sequence Evolution Simulator See code in Appendix A

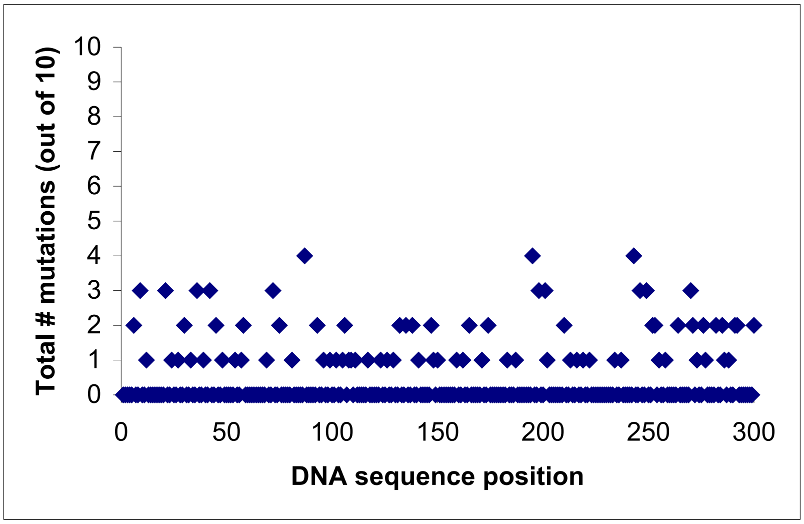
SAMPLE SCATTER PLOTS

(A) No selective constraint



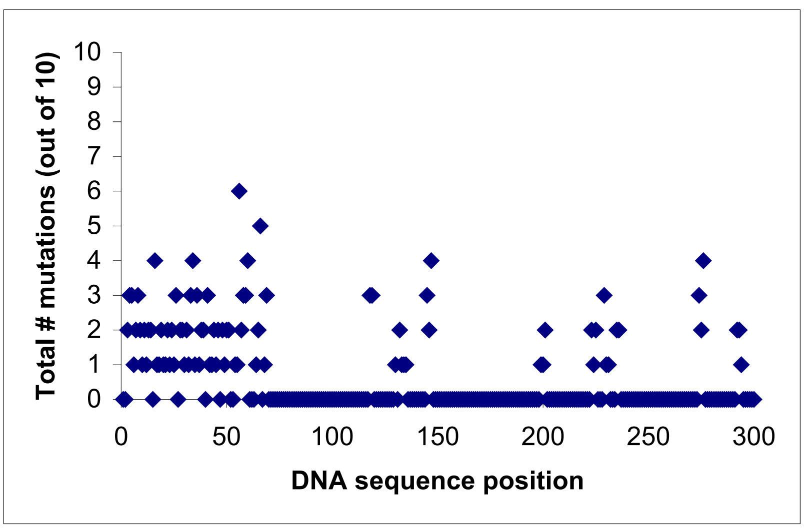
All mutations are allowed in this simulation, and we expect them to occur uniformly throughout the DNA sequence.

(B) Select for 100% amino acid identity to the original sequence

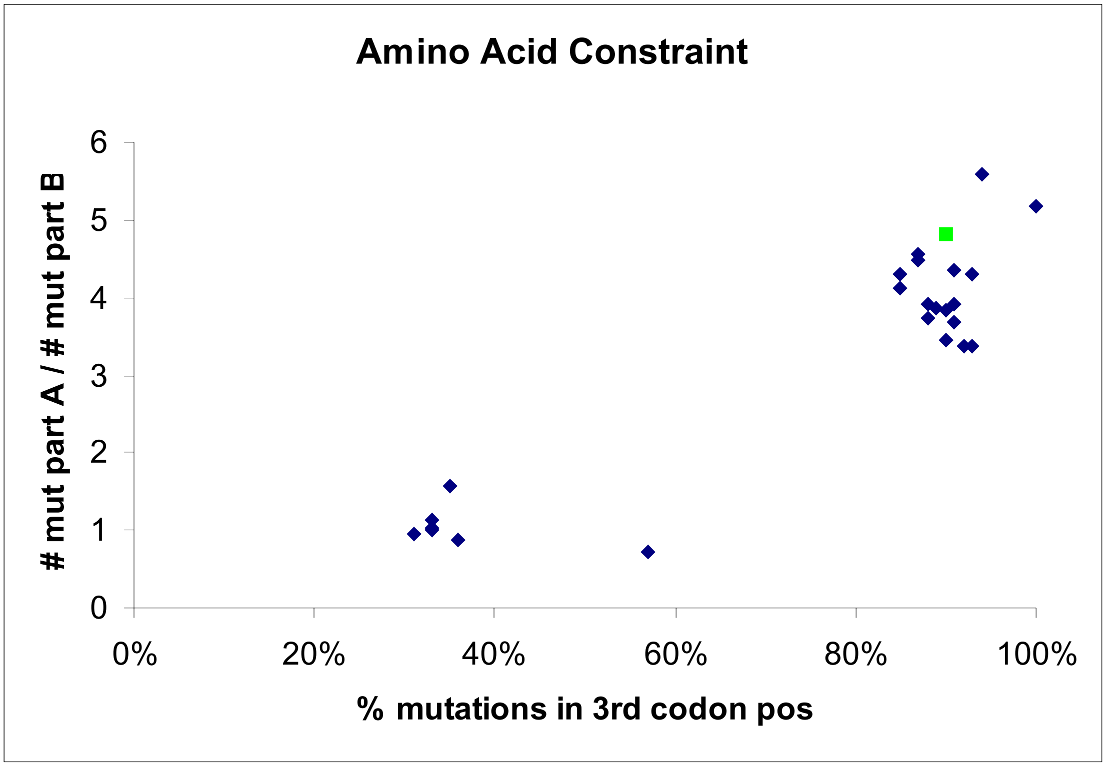


The total number of mutations observed is 3 to 5 fold lower, as a result of a selective constraint for protein sequence identity. Mutations are preferred in degenerate positions of codons (usually the third position of each codon). Degenerate positions are those where nucleotide substitutions do not affect the amino acid identity.

(C) **Select for solvent accessibility scores greater than 150**



The total number of mutations is also lower than in the unconstrained simulation (part A). The mutation rate varies along the nucleotide sequence: mutations are preferred within the first 70 nucleotides, as well as in confined internal "islands." Regions with high mutation rates tend to be close together, which reflects the fact that adjacent amino acids tend to have similar solvent accessibility scores.



We expect that the vast majority of positions to occur in the third position of codons, due to codon degeneracy. This is shown along the horizontal axis. In addition, we expect that the total number of mutations observed should be much higher in part A (no constraint) than in part B (amino acid constraint). This ratio is plotted along the vertical axis.

Each point in this chart represents the results for one student in the class (the green point represents the GSI's answer). Points in the top-right agree with the expected results; points on the bottom probably represent results generated by code with errors.

6. 1 point *extra credit*
What is the gene in question 5?

The gene is YBR009C from the yeast, *Saccharomyces cerevisiae*. It encodes histone H4.

Appendix A) Code for Sequence Evolution Simulator

```
#!/usr/local/bin/perl -w
#
#  FILE: sequenceMutator.pl
#  Simulates a simple process of mutation and selection on a single sequence
#
#  Written by:     Derek Chiang  (derek@rana6.lbl.gov)
#  Date created:   05 Jan 2003
#  Last modified:  06 Jan 2003

use POSIX;
use strict;

if ( @ARGV < 2 )
{
    die "Usage: perl sequenceMutator.pl <Number of generations> <Number of trials>\n";
}

MAIN:
{
    my $codonFile = "codon_table.txt";     # Input file for codon translation
    my $seqFile = "YBR009C.fasta";         # Input file of original sequence
    my $scoreFile = "YBR009C_scores.txt";  # Score file for part (C)
    my $rate = 0.0002;                     # Mutation rate

                                           # Codon table
    my $hrCodons = loadCodons( $codonFile );
    my $origDNA = loadDNA( $seqFile );     # Original DNA sequence
    my $arPosScore = loadScores( $scoreFile );

    my $sequence = $origDNA;               # DNA sequence to mutate

    my @mutationFreq;                      # Expected number of mutations

    print STDERR length( $origDNA ), " nt in original DNA\n";


    for my $trial ( 1 .. $ARGV[1] )
    {
       my $numChanges = 0;                # Total mutations for this trial
       $sequence = $origDNA;
                                           # Evolve sequences
       for my $iter ( 1 .. $ARGV[0] )
       {
           ( $sequence, $numChanges ) = mutateSeq( \@mutationFreq, $sequence, $rate,
$numChanges, $hrCodons, $arPosScore );
                                          # Combines mutation and selection
                                          # Depends on chooseNT() for mutation
                                          # and scoreChange() for selection
       }
                                          # DEBUG statement
       print STDERR "TRIAL ", $trial, ": $numChanges changes\n";
    }

                                           # Formatted output
    printSummary( \@mutationFreq, length( $origDNA ) - 1 );
}
```

```
#------------------------(   loadCodons   )-----------------------#
#  Load translation table from file                               #
#-----------------------------------------------------------------#
sub chooseNT
{
    my ( $nt,
        $arAlphabet ) = @_;
    my @diffNT;                           # Array of nt not identical to $nt
    my $chosenNT;                         # Randomly chosen nt that is not
                                          # identical to $nt

    for my $b ( @$arAlphabet )            # Loop through A,C,G,T
    {
                                         # Add base to array if NOT identical
                                         # to current base
      push( @diffNT, $b ) if ( $b ne $nt );
    }

    $chosenNT = $diffNT[floor(rand(3)/1)]; # Use a random number to pick nt

    return $chosenNT;
}


#------------------------(   loadCodons   )-----------------------#
#  Load translation table from file                               #
#-----------------------------------------------------------------#
sub loadCodons
{
    my ( $fileName ) = @_;
    my %codons;                           # KEY: DNA triplet, VAL: Amino acid

    open( DATA, $fileName) or die "ERROR: Cannot open codong table\n";

    while ( my $line = <DATA> )           # Loop through input file
    {
      chomp $line;
      my @row = split( "\t", $line );
      $codons{$row[0]} = $row[1];
    }
    return ( \%codons );
}


#-------------------------(   loadDNA   )-------------------------#
#  Load DNA sequence from file                                    #
#-----------------------------------------------------------------#
sub loadDNA
{
    my ( $file ) = @_;
    my $sequence;
    open( SEQ, $file ) or die "ERROR: Cannot open sequence file\n";

    while ( my $line = <SEQ> )            # Loop through input file
    {
      chomp $line;
      $sequence .= $line if ( substr($line, 0, 1 ) ne ">" );
    }

    return $sequence;
}
```

```
#------------------------(   loadScores   )-----------------------#
#  Load scores for part C from file                               #
#-----------------------------------------------------------------#
sub loadScores
{
    my ( $file ) = @_;
    my @positionScores;                     # INDEX: Pos-1, VAL: Score

    open( DATA, $file ) or die "ERROR: Cannot open score file\n";
    while ( my $line = <DATA> )             # Loop through input file
    {
       chomp $line;
       my @row = split( "\t", $line );
                                           # Add next score onto the array
       push( @positionScores, $row[1] ) if ( $row[1] =~ /\d+/ );
    }
    return ( \@positionScores );
}


#------------------------(   mutateSeq   )------------------------#
#  MAIN ALGORITHM                                   #
#-----------------------------------------------------------------#
sub mutateSeq
{
    my ( $arMutations,
        $sequence,
        $rate,
        $numChanges,
        $hrCodons,
        $arScores ) =  @_;                # Cumulative sum of mutations

    my @alphabet = ( "A", "C", "G", "T" ); # DNA nucleotide possibilities
    my @nt = split( "", $sequence );
    for my $pos ( 0 .. $#nt )
    {
       my $ran = rand();                   # Generate a random number
                                           # Mutation occurs according to
       if ( $ran < $rate )                 # random process
       {
           my $randomNT = chooseNT( $nt[$pos], \@alphabet );

                                           # Comment out the undesired scoring
                                           # functions below (ALL for part A)
#          if ( scoreChangeB( $sequence, $randomNT, $pos, $hrCodons ) )
#          if ( scoreChangeC( $pos, $arScores ) )
           {
              $nt[$pos] = $randomNT;
              $arMutations->[$pos]++;
              $numChanges++;
           }
       }
    }
    $sequence = join( "", @nt );            # Re-assign seq to include mutation
    return ( $sequence, $numChanges );
}
```

```
#-----------------------(   printSummary   )------------------------#
#  Print a tab-delimited file with column 1 as DNA position         #
#  and column 2 as total number of changes at that position         #
#-------------------------------------------------------------------#
sub printSummary
{
    my ( $arMutnFreq,
        $size ) = @_;

    for my $pos ( 0 .. $size )             # Loop through all DNA positions
    {
       print $pos+1, "\t";
       if ( defined $arMutnFreq->[$pos] )
       {
           print $arMutnFreq->[$pos];     # Print total number of changes
       }
       else
       {
           print "0";                     # No changes observed
       }
       print "\n";
    }
}


#-----------------------(   scoreChangeB   )------------------------#
#  Scoring function for part B                                      #
#  Keep new nt only if the amino acid remains idential to original  #
#-------------------------------------------------------------------#
sub scoreChangeB
{
    my ( $sequence,
        $newNT,
        $pos,
        $hrCodons ) = @_;
    my $frame = $pos % 3;
    my $codonStart = $pos - $frame;

    my $oldCodon = substr( $sequence, $codonStart, 3 );
    my @nt = split( "", $oldCodon );
    $nt[$frame] = $newNT;
    my $newCodon = join( "", @nt );

    return 1 if ( $hrCodons->{$oldCodon} eq $hrCodons->{$newCodon} );
}


#-----------------------(   scoreChangeC   )------------------------#
#  Scoring function for part C                                      #
#  Keep new nt if corrseponding amino acid has score > 150 in the   #
#  file YBR009C_scores.txt                                          #
#-------------------------------------------------------------------#
sub scoreChangeC
{
    my ( $pos,
        $arScores ) = @_;
    my $aaPos = floor( $pos/3 );

    return 1 if ( $arScores->[$aaPos] > 150 );
}
```

```
#-------------------------(   translate   )-------------------------#
#  Translate DNA -> Protein using universal code                    #
#-------------------------------------------------------------------#
sub translate
{
    my ( $dna,
        $hrCodon ) = @_;
    my $protein;

    if ( length( $dna ) % 3 != 0 )          # Error-checking for triplets
    {
      print STDERR "ERROR: DNA sequence is not of triplet length!\n";
      return;
    }

    for my $c ( 0 .. length($dna)/3 - 1 )   # Loop through codons
    {
      my $codon = substr( $dna, 3*$c, 3 );# Extract next DNA triplet
      $protein .= $hrCodon->{$codon};     # Add next amino acid to protein
    }

    return $protein;
}
```